Package 'GHAgricProductivityLab'

<div align="center">November 16, 2025</div>

Type Package

Title Agricultural Productivity in Ghana

Version 0.0.0.9000

Author Francis Tsiboe [aut, cre] (<https://orcid.org/0000-0001-5984-1072>)

Maintainer Francis Tsiboe <ftsiboe@hotmail.com>

Contributor -

Reviewer -

Creator Francis Tsiboe

Description Provides tools and datasets for investigating the drivers of agricultural production shortfalls in Ghana.
It compiles research studies that examine farmer-specific and institutional factors to assess whether inefficiencies arise from technical inefficiency, technology gaps, or both. The package offers empirical evidence to inform policy discussions and interventions aimed at improving agricultural productivity, particularly in contexts with limited access to modern technologies.

License GPL-3 + file LICENSE

URL <https://github.com/ftsiboe/GHAgricProductivityLab>

BugReports <https://github.com/ftsiboe/GHAgricProductivityLab/issues>

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports MatchIt, data.table, haven, stats, sfaR, piggyback

Remotes github::dylan-turner25/rfcip

Suggests knitr, crayon, tidyr, withr, rmarkdown, lavaan, quadprog, emdbook, cobalt, cli, dplyr, doBy, stringr, MASS, Matrix, corpcor, testthat (>= 3.0.0), car, frontier, micEcon, rgenoud, purrr, randomForest, CBPS, dbarts, optmatch, Matching

LazyData true

Cite-us If you find it useful, please consider staring the repository and citing the following studies

- Tsiboe F. (2021). Chronic sources of low cocoa production in Ghana, new insights from meta-analysis of old survey data.
Agricultural and Resource Economics Review.50(2) 226-251.
- Tsiboe, F., Egyir, I. S., & Anaman, G. (2021). Effect of fertilizer subsidy on household level cereal production in Ghana.
Scientific African, 13, e00916.
- Tsiboe F. (2022). Nationally Representative Farm/Household Level Dataset on Crop Production in Ghana from 1987-2017.

Contents

covariate_balance	*Compute Covariate Balance Summaries Across Matching Specs*

Description

Compute Covariate Balance Summaries Across Matching Specs

Usage

```
covariate_balance(match_specifications, matching_output_directory)
```

Arguments

match_specifications

data.frame/data.table with at least columns: boot, ARRAY, and the spec fields stored in RDS (e.g., method, distance, link).

matching_output_directory

Directory containing RDS files named as "0001.rds", "0002.rds", etc.

Details

Reads each matching result RDS (expected to contain `m.out` and `match_specifications`), extracts balance via `cobalt::bal.tab`, reshapes, computes a composite balance $rate = mean((Diff - 0)^2, (KS - 0)^2, (V_Ratio - 1)^2)$, and averages by `ARRAY`, `method`, `distance`, `link`, `sample`.

Value

A list with:

rate	data.frame of mean balance metrics by spec (Adj sample only) with a composite `rate`.
bal_tab	long-format balance table per covariate/stat/sample/spec.

See Also

Other matching: `draw_matched_samples()`, `match_sample_specifications()`, `treatment_effect_calculation()`, `treatment_effect_summary()`, `write_match_formulas()`

---

draw_matched_samples   *Draw matched samples (stratified bootstrap + matching)*

Description

Performs stratified resampling at the `Surveyx`, `EaId` level (excluding EaIds listed for the current bootstrap `ID`) and computes adjusted sampling weights used for matching. Then fits a matching model per `match_specifications[i, ]` using **MatchIt**, with exact matching on `Emch` and distance/link from `match_specifications`.

Usage

```
draw_matched_samples(
  i,
  data,
  match_variables_exact,
  match_variables_scaler,
  match_variables_factor,
  match_specifications,
  sample_draw_list,
  verbose = FALSE
)
```

Arguments

i	Integer index selecting the row of `match_specifications` to use.
data	A data.frame/data.table containing at least: `Surveyx`, `EaId`, `HhId`, `Mid`, `unique_identifier`, `Weight`, `Treat`, plus columns named in `Emch`, `Scle`, `Fixd`.
match_variables_exact	
	A character vector of variable names to be included in the exact match component (e.g., `"gender"`, `"region"`).

match_variables_scaler

> A character vector of continuous or scalar variable names to include as covariates in the general match formula (e.g., "age", "income").

match_variables_factor

> A character vector of factor variable names to be included as dummy-coded categorical covariates in the general match formula.

match_specifications

> Data frame of matching specifications with columns boot, method, distance, and optionally link.

sample_draw_list

> Data frame where column ID identifies the bootstrap draw, and each remaining column corresponds to a Survey containing the sampled EaId.

verbose   Logical; if TRUE, prints the chosen matching method and timing. Default FALSE.

Details

Adjusted weights are computed as $pWeight = Weight \times (alloc/allocj)$, where alloc is the pre-exclusion sum of Weight by Surveyx, EaId and allocj is the post-exclusion sum.

Exact matching is performed on variables in Emch. The distance model formula is constructed as Treat ~ Scle + Fixd. When distance == "glm", m.order = "largest" is used; otherwise "closest".

Value

A list with:

match_specifications The selected row from match_specifications.

m.out The matchit object.

md Matched data: Surveyx, EaId, HhId, Mid, unique_identifier, weights, pWeight.

df The analysis data with adjusted weights: Surveyx, EaId, HhId, Mid, unique_identifier, pWeight.

See Also

Other matching: covariate_balance(), match_sample_specifications(), treatment_effect_calculation(), treatment_effect_summary(), write_match_formulas()

---

draw_msf_estimations   *Run a single MSF draw and summarize stochastic frontier results*

Description

Performs one iteration of multi-stage stochastic frontier (MSF) estimation for a given draw. The function (i) filters the analysis sample using a draw-specific exclusion list, (ii) calls msf_workhorse() to estimate production, inefficiency, and risk components by survey, and (iii) optionally builds disaggregated efficiency score summaries.

Usage

```
draw_msf_estimations(
  draw,
  drawlist,
  surveyy = FALSE,
  data,
  output_variable,
  input_variables,
  inefficiency_covariates = NULL,
  risk_covariates = NULL,
  weight_variable = NULL,
  production_slope_shifters = NULL,
  intercept_shifters = NULL,
  f,
  d,
  identifiers,
  include_trend = FALSE,
  technology_variable = NULL,
  matching_type = NULL,
  adoption_covariates = NULL,
  intercept_shifters_meta = NULL,
  disagscors_list = NULL
)
```

Arguments

draw	Integer (or numeric coercible to integer). Draw iteration index. Typically `0` is used for the full baseline sample and positive integers for resampled or matched draws.
drawlist	A data.frame or similar object containing exclusion information by draw. Rows correspond to draws, with column `ID` identifying the draw and subsequent columns containing `EaId` values to remove from `data` for that draw.
surveyy	Logical; if `TRUE`, uses the survey labels stored in `data$Surveyx`. If `FALSE` (default), all observations are assigned to a synthetic survey labeled `"GLSS0"` for estimation.
data	A data.frame or data.table containing the estimation sample. Must include, at minimum, the columns referenced in other arguments (e.g., `output_variable`, `input_variables`, `identifiers`, `weight_variable`, technology and matching variables, and any variables used in `disagscors_list`).
output_variable	Character scalar. Name of the dependent (output) variable used in the production frontier.
input_variables	Character vector of input (production) variables entering the stochastic frontier.
inefficiency_covariates	Optional named list specifying variables in the inefficiency function. Typical structure: `list(Svarlist = c(...), Fvarlist = c(...))`.
risk_covariates	Optional named list specifying variables in the production risk (noise) function. If `NULL`, a homoskedastic noise term is usually implied.

weight_variable

> Optional character scalar giving the name of the sampling weight variable in data. Used when computing weighted summaries (e.g., weighted means) of efficiency scores.

production_slope_shifters

> Character scalar naming a variable that shifts the production-function slope (e.g., technology shifter). Defaults to `NULL` to indicate no slope shifter.

intercept_shifters

> Optional named list of intercept shifter variables for the baseline (unmatched) sample. Typical structure: `list(Svarlist = c(...), Fvarlist = c(...))`.

f
> Functional form identifier passed to `msf_workhorse()` (e.g., Cobb-Douglas, translog). The exact meaning is handled by the workhorse function.

d
> Distributional form identifier for the inefficiency term (e.g., half-normal, truncated-normal), passed through to `msf_workhorse()`.

identifiers
> Character vector of variable names that uniquely identify units (e.g., household, plot, or observation IDs). These are used to merge efficiency scores back to data and for disaggregated summaries.

include_trend
> Logical; if `TRUE`, includes a technology trend variable (given by `technology_variable`) in the frontier. Defaults to `FALSE`.

technology_variable

> Optional character scalar naming the technology (trend) variable to be used when `include_trend = TRUE`.

matching_type
> Optional variable or object controlling nearest-neighbor matching; passed to `msf_workhorse()`. The expected type and structure depend on the implementation of that workhorse function.

adoption_covariates

> Optional named list specifying inefficiency-function variables for the matched sample (post-matching specification). Same structure as `inefficiency_covariates`.

intercept_shifters_meta

> Optional named list of intercept shifters for the matched sample. Same structure as `intercept_shifters`.

disagscors_list

> Optional character vector of variable names for which disaggregated efficiency score summaries should be computed. For each variable in this list, the function builds weighted and unweighted summaries of TE/TE0/TGR/MTE by survey, sample, and disaggregation level.

Details

The main outputs include model estimates, mean/percentile summaries, and (optionally) disaggregated efficiency statistics by user-specified grouping variables. For draws other than zero, sample-level results are dropped to reduce storage.

1. **Draw-specific filtering:** Using `drawlist`, the function removes any EaId values associated with the current draw from data. This allows for bootstrap, jackknife, or matched-sample resampling.

2. **Survey-specific estimation:** After setting the Surveyy label (either from Surveyx or collapsed to `"GLSS0"`), the function loops over unique survey labels and calls `msf_workhorse()` for each. The workhorse is expected to return a list with components such as `sf_estm`, `el_mean`, `ef_mean`, `rk_mean`, `ef_dist`, `rk_dist`, `el_samp`, `ef_samp`, and `rk_samp`.

3. **Disaggregated efficiency summaries:** If `disagscors_list` is provided, the function constructs weighted and unweighted efficiency statistics (e.g., weighted mean, mean, median, mode) for TE/TE0/TGR/MTE by survey, sample, restriction status, technology, and disaggregation level. Results are stored in `res$disagscors`.

4. **Draw-specific pruning:** When `draw != 0`, sample-level objects (`el_samp`, `ef_samp`, `rk_samp`) are removed from the returned list to reduce memory usage, keeping only aggregate results.

Internally, the function uses **data.table**, **dplyr**, **tidyr**, **doBy**, and **crayon** for data manipulation and progress messages. All estimation is delegated to `msf_workhorse()`.

Value

A named list with components:

- `sf_estm` - Combined frontier parameter estimates across surveys (rows tagged with `Surveyy` and `draw`).
- `el_mean`, `ef_mean`, `rk_mean` - Mean/summary statistics for elasticities, efficiency, and risk metrics.
- `ef_dist`, `rk_dist` - Distributions of efficiency and risk quantities.
- `el_samp`, `ef_samp`, `rk_samp` - Sample-level results (only retained when `draw == 0`).
- `disagscors` - Optional disaggregated efficiency summary table if `disagscors_list` is not `NULL`; otherwise `NULL`.

If an error occurs anywhere in the pipeline, the function returns `NULL`.

---

draw_msf_summary   *Summarize multi-draw Meta Stochastic Frontier (MSF) results*

Description

Aggregates results from multiple stochastic frontier analysis (SFA) / MSF draws (e.g., from a bootstrap or jackknife procedure) and computes jackknife-style summary statistics for coefficients, efficiency scores, elasticities, risk measures, and their distributions.

Usage

```
draw_msf_summary(res, technology_legend)
```

Arguments

res   List of draw-level result objects. Each element is expected to be a list with (at least) the following components:

- `sf_estm`: coefficient-level estimates and diagnostics for naive, group, and meta-frontiers, including a `draw` index.
- `ef_mean`: aggregated efficiency statistics (e.g., TE0, TE, TGR, MTE) by sample, technology, survey, and estimation type.
- `el_mean`: aggregated elasticity statistics by input, technology, and survey.
- `rk_mean`: aggregated risk statistics, if available.
- `ef_dist`: histogram-style efficiency distributions (counts/weights) over value ranges (range, `Frqlevel`).

- `rk_dist`: histogram-style risk distributions, if available.
- `disagscors`: optional disaggregated efficiency summaries by subgroup levels.
- `el_samp`, `ef_samp`, `rk_samp`: observation-level elasticities, efficiencies, and risk measures for at least the baseline draw (usually `draw == 0`).

Each component must include a `draw` column that distinguishes the baseline draw (typically `draw == 0`) from resampled draws (`draw != 0`).

technology_legend

Data.frame providing the mapping between numeric technology codes and their labels, typically with at least:

- `Tech`: numeric technology index used internally in MSF estimation, and
- a corresponding label column (e.g., the original `technology_variable`).

This is used to compute technology gaps (e.g., efficiency or disaggregated efficiency gaps) relative to the reference technology (smallest value in `technology_legend$Tech`).

Details

The function assumes each element of `res` is the output of a single draw from an MSF pipeline (e.g., `msf_workhorse()` followed by `draw_msf_estimations()`), containing components such as: `sf_estm`, `ef_mean`, `el_mean`, `rk_mean`, `ef_dist`, `rk_dist`, and (optionally) `disagscors`.

For each component, the function:

1. Stacks all draws using `data.table::rbindlist()`.
2. Drops non-finite values (`NA`, `Inf`, `-Inf`, `NaN`).
3. For each statistic (e.g., coefficient estimate, mean efficiency), identifies the baseline draw (`draw == 0`) and joins it with resampled draws (`draw != 0`) aggregated via `doBy::summaryBy()` to compute:
   - `Estimate.mean`: mean across non-baseline draws,
   - `Estimate.sd`: standard deviation across non-baseline draws,
   - `Estimate.length`: number of non-baseline draws.
4. Computes a jackknife-style test statistic and p-value:
   - `jack_zv` = `Estimate / Estimate.sd`,
   - `jack_pv` = `2 * (1 - pt(|jack_zv|, df = Estimate.length))`,

   where `Estimate` is the baseline draw statistic and the distribution of resampled draws is treated as a reference distribution.

For disaggregated scores (`disagscors`), the function also:

- Computes technology-specific disaggregated efficiency gaps (level and percent) relative to the reference technology in `technology_legend`, and
- Aggregates these gaps across draws in the same jackknife fashion as for the main efficiency statistics.

Value

A list with the following components:

- `sf_estm`: Data.frame of jackknife summary statistics for frontier coefficients and diagnostics across draws, with columns such as `Estimate`, `Estimate.mean`, `Estimate.sd`, `Estimate.length`, `jack_zv`, and `jack_pv`, indexed by `Survey`, `CoefName`, `Tech`, `sample`, and `restrict`.

- `ef_mean`: Jackknife summaries for efficiency statistics (TE0, TE, TGR, MTE), by sample, technology, survey, type, estimation type, statistic, and restriction.

- `el_mean`: Jackknife summaries for elasticity statistics, by input, technology, survey, statistic, and restriction.

- `rk_mean`: (If available) jackknife summaries for risk statistics, with type = "risk".

- `ef_dist`: Jackknife summaries of efficiency distributions (counts and weights) over ranges, with stat indicating whether the row refers to counts or weights.

- `rk_dist`: (If available) jackknife summaries of risk distributions over ranges, with type = "risk".

- `disagscors`: (If available) jackknife summaries for disaggregated efficiency statistics and their gaps by subgroup level and technology.

- `el_samp`: Observation-level elasticity outputs from the first draw in res (typically the baseline draw).

- `ef_samp`: Observation-level efficiency scores from the first draw in res.

- `rk_samp`: Observation-level risk measures from the first draw in res.

---

equation_editor	*Build production, inefficiency, and risk equations for SFA/MSF*

Description

Constructs formula objects for the production function, the inefficiency function, and the production risk function used in stochastic frontier and meta-stochastic frontier estimations. The function takes a generic functional form (FXN) and conditionally augments it with intercept and slope shifters, as well as covariates for the inefficiency (u) and noise (v) equations.

Usage

```
equation_editor(
  data,
  FXN,
  outcome,
  production_slope_shifters = NULL,
  intercept_shifters = NULL,
  inefficiency_covariates = NULL,
  risk_covariates = NULL
)
```

Arguments

data	Data.frame or similar object containing all variables referenced in FXN, outcome, production_slope_shifters, intercept_shifters, inefficiency_covariates, and risk_covariates.
FXN	Functional form of the production function. Typically a named object where FXN[[1]] is a character string representing the production frontier in terms of input variables (e.g., "lnI1 + lnI2 + I(1/2 * lnI1^2) + lnI1:lnI2" for a translog).

outcome   Character scalar giving the dependent variable name for the production equation
                         (e.g., "Y" or "lnY").

production_slope_shifters

                         Character scalar naming a variable used as a slope shifter in the production func-
                         tion. If equal to NULL (default), no slope shifter is applied. Otherwise, the pro-
                         duction equation is specified as an interaction between factor(production_slope_shifters)
                         and FXN[[1]], net of the baseline FXN[[1]].

intercept_shifters

                         Optional list of intercept shifters for the production function with elements:

                         • scalar_variables: character vector of continuous shifter variables.
                         • factor_variables: character vector of categorical shifter variables (in-
                           cluded as factor() terms).

                         If intercept_shifters is NULL or both components are NULL, the production
                         function only uses FXN[[1]].

inefficiency_covariates

                         Optional list of covariates for the inefficiency (u) equation with elements:

                         • scalar_variables: character vector of continuous covariates.
                         • factor_variables: character vector of categorical covariates (included as
                           factor() terms).

                         If inefficiency_covariates is NULL or both components are NULL, the ineffi-
                         ciency equation is set to NULL.

risk_covariates

                         Optional list of covariates for the production risk (v) equation with elements:

                         • scalar_variables: character vector of continuous covariates.
                         • factor_variables: character vector of categorical covariates (included as
                           factor() terms).

                         If risk_covariates is NULL or both components are NULL, the risk equation is
                         set to NULL.

Details

Continuous covariates are only included when their coefficient of variation exceeds a small thresh-
old (|CV| > 0.001), and categorical covariates are included only if they have more than one level and
each level has at least a minimal share of the sample (>0.01% of observations). This helps avoid
numerical issues from nearly-constant or extremely sparse regressors.

The function builds three formula objects:

Production function (prodfxn) Starts from outcome ~ FXN[[1]]. If intercept shifters are sup-
    plied, they are appended as additive terms (continuous variables directly, categorical variables
    as factor()). When a production_slope_shifters is specified (not NULL), the main pro-
    duction part is reparameterized as:

        outcome ~ factor(production_slope_shifters) * (FXN[[1]]) - (FXN[[1]]) + shifters

Inefficiency function (uequ) Built as ~ 1 + ... using continuous and/or categorical variables spec-
    ified in inefficiency_covariates. If neither scalar_variables nor factor_variables
    are supplied (or all are filtered out), uequ is set to NULL.

Risk function (vequ) Built analogously to uequ using risk_covariates. If no valid covariates
    remain, vequ is set to NULL.

In all three parts, continuous and categorical candidate variables are screened to avoid near-constant or extremely sparse regressors:

- Continuous variables are retained only if $|sd(x)/mean(x)| > 0.001$.
- Categorical variables are retained only if they have more than one level and the smallest category represents at least 0.01% of the sample.

Value

A list with three components:

- prodfxn: a [formula](#) for the production frontier.
- uequ: a formula for the inefficiency equation, or NULL if no inefficiency covariates were retained.
- vequ: a formula for the production risk equation, or NULL if no risk covariates were retained.

See Also

Other frontier analysis: [fit_organizer()](#), [msf_workhorse()](#), [sf_functional_forms()](#), [sf_model_specifications](#) [sf_workhorse()](#), [sfaR_summary()](#)

---

fit_organizer   *Organize frontier coefficients and variance-covariance matrix*

Description

Extracts and structures the coefficients and variance-covariance matrix from a fitted stochastic frontier (or related) model, mapping them into a generic parameterization compatible with flexible functional forms such as Cobb-Douglas, translog, quadratic, and linear frontiers.

Usage

```
fit_organizer(fit, number_of_inputs, FXN)
```

Arguments

fit
: Fitted model object (typically the result of a call to sfacross() or a similar frontier regression). The object must support coef() and vcov() methods.

number_of_inputs
: Integer. Number of input variables in the production function. This determines how many first-order and second-order terms are expected.

FXN
: Named object describing the functional form of the production function. The name (i.e., names(FXN)) is used to branch between supported forms:

- ʺTLʺ: translog.
- ʺQDʺ: quadratic.
- ʺCDʺ: Cobb-Douglas.
- ʺLNʺ: linear.

For ʺTLʺ and ʺCDʺ, inputs are expected to enter in logged form as ʺlnI1ʺ, ʺlnI2ʺ, ...; for ʺQDʺ and ʺLNʺ, in levels as ʺI1ʺ, ʺI2ʺ, ...

Details

For translog and quadratic specifications, the function identifies and orders both first-order and second-order (interaction) terms. For Cobb-Douglas and linear forms, it augments the estimated coefficients with zero-valued second-order terms and builds a conformable variance-covariance matrix (filled with zeros for the non-estimated interactions), so that downstream routines can work with a common parameter vector and matrix.

The function constructs three key objects:

- **est_list**: character vector of coefficient names as they appear in the fitted model object (e.g., $"(Intercept)"$, $"lnI1"$, $"lnI2"$, interaction terms such as $"I(1/2 * lnI1 * lnI1)"$ or $"lnI1:lnI2"$, etc.).

- **est_name**: generic parameter labels corresponding to the structural coefficients, such as $"a\_0"$, $"a\_1"$, ... for first-order terms and $"b\_i\_j"$ for second-order terms.

- **est_coef** and **est_vcov**: numeric vector of coefficients and conformable variance-covariance matrix, reordered and expanded (for CD/LN) to match the generic parameterization in est_name.

For Cobb-Douglas and linear specifications ($"CD"$, $"LN"$), only first-order terms are typically estimated directly. The function therefore:

1. Copies the estimated coefficients for first-order terms.

2. Appends zeros for all second-order coefficients so that the length matches the full parameterization implied by number_of_inputs.

3. Builds a larger variance-covariance matrix with the original submatrix in the upper-left corner and zeros elsewhere.

This unified representation is useful for elasticities, curvature checks, and minimum-distance procedures that require a full set of parameters and a matching variance-covariance matrix regardless of the underlying functional form.

Value

A list with components:

- est_coef: Numeric named vector of organized coefficients. Names follow the generic parameter labels a_0, a_1, ..., b_i_j.

- est_vcov: Square variance-covariance matrix corresponding to est_coef, with row/column names matching the coefficient names.

- est_list: Character vector of coefficient names as they were extracted from fit (i.e., the original model coefficient labels used to build est_coef and est_vcov).

See Also

Other frontier analysis: equation_editor(), msf_workhorse(), sf_functional_forms(), sf_model_specification sf_workhorse(), sfaR_summary()

---

get_crop_area_list   *Extract Matching Crop Area Columns from a Dataset*

Description

Identifies and returns the column names in a dataset corresponding to crop area variables for a specified set of crops. The function looks for column names that start with "Area_" and match any of the crops provided in selected_crops.

Usage

```
get_crop_area_list(
  data,
  selected_crops = c("Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra",
    "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", "Yam")
)
```

Arguments

data	A data.frame or data.table containing crop-related variables. Column names are expected to include fields prefixed with "Area_", such as "Area_Maize".
selected_crops	A character vector specifying the crop names to filter. Defaults to a common set of crops including "Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra", "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", and "Yam".

Value

A character vector containing the names of columns in data that correspond to the specified crop area variables (e.g., "Area_Maize", "Area_Rice"). Returns an empty vector if no matching columns are found.

---

get_household_data   *Download and Load Harmonized Household Data from the GHA-gricProductivityLab Repository*

Description

Retrieves a harmonized household- or farm-level dataset from the **GHAgricProductivityLab** GitHub repository using **piggyback**, stores it in a package-specific cache directory, and returns the dataset as a data.frame. The file is downloaded only once and reused from the local cache on future calls.

Usage

```
get_household_data(
  dataset = "harmonized_crop_farmer_data",
  github_token = NULL
)
```

Arguments

dataset   Character string. Base name of the dataset to retrieve (without file extension).
                    Must correspond to a .dta file in the hh_data GitHub release (e.g., "harmonized_crop_farmer_dat

github_token   Optional GitHub personal access token (PAT). If NULL, the function checks the
                    environment variable GHProdLab_TOKEN. If that is also missing, the piggyback
                    download will use default authentication behavior.

Details

The function downloads a Stata .dta file associated with the chosen dataset from the GitHub release
labeled hh_data. It uses the package-specific cache directory determined by:

```
tools::R_user_dir("GHAgricProductivityLab", which = "cache")
```

If the file already exists locally, it is not re-downloaded.

GitHub Authentication

- If github_token is supplied, it is used.

- Otherwise, the function looks for environment variable GHProdLab_TOKEN.

- If neither is available, the function falls back to default GitHub credentials (e.g., from gh CLI
  or cached credentials).

Value

A data.frame containing the requested harmonized dataset.

---

harmonized_data_prep   *Prepare Data for Agricultural Productivity Analysis*

Description

Cleans and transforms a dataset by creating new variables, applying log transformations, converting
selected variables to factors or characters, and recoding education levels. The function is designed
to standardize inputs for further analysis of agricultural productivity.

Usage

```
harmonized_data_prep(data)
```

Arguments

data   A data.frame or data.table containing household- and farm-level variables
                    such as weights, demographic information, and agricultural inputs.

Value

A cleaned and transformed data.frame or data.table with additional variables ready for analysis.

match_sample_specifications
Draw / Match Sample Specifications

Description

Generates (i) a draw list by sampling `EaId` within each unique `Survey` group and (ii) a grid of matching specifications for each bootstrap draw.

Usage

```
match_sample_specifications(number_of_draws = NULL, data, myseed = NULL)
```

Arguments

number_of_draws

Integer. The number of draws to perform per `Survey`.

data   A data.frame or data.table containing at least the columns `Survey` and `EaId`.

myseed   Integer. Seed for random number generation (default `03242025`).

Details

Draw list: For each unique value of `Survey`, the function samples `number_of_draws` `EaId` values with replacement and prepends a 0 row (ID = 0) for the baseline. The result is then spread to wide format with one column per `Survey`.

Matching specs: For each draw ID, creates a set of matching model specifications that include:

- Nearest neighbor with distances: `"euclidean"`, `"scaled_euclidean"`, `"mahalanobis"`, `"robust_mahalanobis"`.
- Nearest neighbor with distance `"glm"` and links: `"logit"`, `"probit"`, `"cloglog"`, `"cauchit"`.

An `ARRAY` index is added for convenience.

Value

A list with two elements:

`m.specs` A data.frame of matching specifications with columns `boot`, `method`, `distance`, `link`, `ARRAY`.

`drawlist` A data.frame in wide format where each `Survey` is a column and rows correspond to draw ID (0:number_of_draws).

Note

Requires that `data` contain `Survey` and `EaId`. `tidyr::spread()` is used for wide reshaping (consider `tidyr::pivot_wider()` in new code).

See Also

Other matching: `covariate_balance()`, `draw_matched_samples()`, `treatment_effect_calculation()`, `treatment_effect_summary()`, `write_match_formulas()`

---

msf_workhorse   *Meta stochastic frontier (MSF) workhorse*

Description

Performs Meta Stochastic Frontier (MSF) analysis in several stages:

1. **Naive SF:** Fits a pooled stochastic frontier and computes baseline technical efficiency (TE) measures.
2. **Group SF:** If a technology variable `technology_variable` is supplied, splits the sample into technology groups and fits group-specific SF models.
3. **Meta SF (unmatched and matched):** Constructs a meta-frontier using fitted values from group models and, optionally, nearest-neighbor matching specifications. From this, it derives technology gap ratios (TGR) and meta-technical efficiency (MTE).
4. **Summaries and distributions:** Produces weighted and unweighted summaries and distributional statistics for efficiency, elasticity, and risk across technologies, samples (unmatched/matched), and surveys.

Usage

```
msf_workhorse(
  data,
  output_variable,
  input_variables,
  inefficiency_covariates = NULL,
  risk_covariates = NULL,
  weight_variable = NULL,
  production_slope_shifters = NULL,
  intercept_shifters = NULL,
  f,
  d,
  identifiers,
  include_trend = FALSE,
  technology_variable = NULL,
  matching_type = NULL,
  adoption_covariates = NULL,
  intercept_shifters_meta = NULL
)
```

Arguments

data	A data.frame or data.table containing the estimation sample. Must include the dependent variable `output_variable`, the inputs in `input_variables`, the weight variable `weight_variable`, unique ID variables in `identifiers`, any inefficiency and risk covariates in `inefficiency_covariates` and `risk_covariates`, the technology variable `technology_variable` (if used), and any variables required by matching objects on disk (e.g., "Surveyx", "EaId", "HhId", "Mid", "unique_identifier").
output_variable	
	Character scalar. Name of the dependent (output) variable used in the production frontier.

input_variables

        Character vector of input variable names (e.g., land, labor, capital) passed through to `sf_workhorse`.

inefficiency_covariates

        Optional named list specifying variables in the inefficiency function for the naive and group frontiers. Typical structure: `list(scalar_variables = c(...),` `factor_variables = c(...))`.

risk_covariates

        Optional named list specifying variables in the production risk (noise) function for the naive and group frontiers. Same structure as `inefficiency_covariates`. If `NULL`, a homoskedastic noise term is usually implied.

weight_variable

        Character scalar. Name of the sampling/observation weight variable in `data`. Observations with zero weight are dropped.

production_slope_shifters

        Character scalar giving the name of a slope-shifter variable in `data` (e.g., technology shifter, policy dummy). Defaults to `NULL` for no slope shifter.

intercept_shifters

        Optional named list of intercept shifter variables for the naive and group frontiers. Typical structure: `list(scalar_variables = c(...), factor_variables` `= c(...))`, passed to `sf_workhorse`.

f   Functional form index for the production frontier (e.g., Cobb-Douglas, translog, quadratic). The index is passed to `sf_workhorse` and ultimately to the internal form-selection utilities (e.g., sf_functional_forms).

d   Distributional form index for the inefficiency term (e.g., half-normal, truncated-normal). Passed to `sf_workhorse`.

identifiers   Character vector of variable names that uniquely identify observations (e.g., `c("unique_identifier","Survey","CropID","HhId","EaId","Mid")`). These IDs are used when merging scores and summarizing by technology or sample.

include_trend   Logical; if `TRUE`, the last element in `input_variables` is treated as a trend/technology variable in the production function. Passed through to `sf_workhorse`. Defaults to `FALSE`.

technology_variable

        Optional character scalar naming the technology variable (e.g., region, system, period) used to define groups for group SF and meta-frontier estimation. If `NULL`, no technology groups are formed and only the naive frontier and TE are computed.

matching_type   Optional character scalar controlling which nearest-neighbor matching specifications to use for meta-frontier estimation. When not `NULL`, the function reads matching specifications and matched samples from `"results/matching/"` and related RDS files. Setting `matching_type = "optimal"` restricts attention to the subset of matching specifications labeled as optimal.

adoption_covariates

        Optional named list specifying inefficiency-function covariates for the meta-frontier (matched/unmatched) estimation. If `NULL`, defaults to `inefficiency_covariates`.

intercept_shifters_meta

        Optional named list of intercept shifters for the meta-frontier estimation. If `NULL`, defaults to `intercept_shifters`.

Details

The function is designed as a high-level orchestrator around `sf_workhorse`, assembling naive, group-level, and meta-frontier results into a unified output structure suitable for MSF analysis.

The workflow can be summarized as:

- **Naive SFA (TE0):** Calls `sf_workhorse` on the full sample to obtain baseline efficiencies (teBC, teJLMS, teMO). If no technology variable is specified (`technology_variable = NULL`), these naive efficiencies are the final scores and only TE-related summaries are produced.

- **Group SFA (TE):** When `technology_variable` is provided, the sample is recoded into numeric technology groups (Tech), and `sf_workhorse` is applied separately to each group. Group-level efficiencies are combined into TE measures by group and survey.

- **Meta-frontier (TGR and MTE):** Using fitted group-frontier values (Yhat) from the restricted models, the function fits meta-frontiers (unmatched and, optionally, matched) again via `sf_workhorse`. Technology gap ratios (TGR) are derived from these meta-frontiers, and meta-technical efficiency (MTE) is computed as TE * TGR.

- **Matching layer (optional):** If `matching_type` is given, the function loads matching specifications (mspecs, mspecs_optimal) and corresponding matched samples from disk (e.g., "results/matching/"), fits additional meta-frontiers by matching specification, and augments the set of scores and summaries with these matched samples.

- **Summaries and distributions:**

    Scores: TE0, TE, TGR, and MTE are reshaped into long form and aggregated to compute weighted/unweighted means, medians, modes, and distribution histograms (both count-based and weight-based) across surveys, samples, technologies, and estimation types. When `technology_variable` is not NULL, the function also computes technology gaps (level and percent) relative to the minimum technology in `technology_legend`.

    Elasticities: Elasticities from the underlying SF models (naive, group, meta) are combined and summarized in a similar way, including technology-gap metrics for elasticities when `technology_variable` is provided.

    Risk: Risk measures derived from sf_workhorse are combined, summarized, and used to build distributional statistics analogous to those for efficiency.

- **LR tests:** When `technology_variable` is not NULL, the function builds likelihood ratio test statistics comparing a naive pooled frontier to the combination of group and meta-frontiers, storing these as rows with CoefName = "LRT" in the `sf_estm` output.

Throughout, the function uses **dplyr**, **tidyr**, **data.table**, **doBy**, and related helper functions for reshaping and summarizing the output of `sf_workhorse`.

Value

A named list with the following components:

- `sf_estm`: Data.frame of coefficient-level summaries from all naive, group, and meta-frontiers (including LR tests), augmented with technology identifiers (Tech) and sample labels (e.g., "unmatched", matching-spec names).

- `ef_mean`: Data.frame of aggregated efficiency statistics (TE0, TE, TGR, MTE), including weighted/unweighted means, medians, and modes, by survey, sample, technology, type, estimation type, and restriction status. When `technology_variable` is supplied, also includes efficiency gap levels and percentages.

- `ef_dist`: Data.frame of efficiency distributions (histogram counts and weights) over efficiency ranges, by survey, sample, technology, type, estimation type, and restriction.

- rk_dist: Data.frame of risk distributions (histogram counts and weights) analogous to ef_dist, with type = "risk".

- el_mean: Data.frame of aggregated elasticity statistics (weighted/unweighted means, medians, modes) by input, survey, sample, technology, and restriction, including elasticity-gap measures when technology_variable is provided.

- rk_mean: Data.frame of aggregated risk statistics (weighted and unweighted) by survey, sample, technology, and restriction, with risk-gap measures when technology_variable is provided.

- el_samp: Data.frame of observation-level elasticities for all models and samples (excluding the synthetic "GLSS0" survey rows).

- ef_samp: Data.frame of observation-level efficiency scores (TE0, TE, TGR, MTE) including IDs, weights, technologies, and sample labels.

- rk_samp: Data.frame of observation-level risk measures for all models and samples (excluding the synthetic "GLSS0" survey rows).

See Also

Other frontier analysis: `equation_editor()`, `fit_organizer()`, `sf_functional_forms()`, `sf_model_specification`, `sf_workhorse()`, `sfaR_summary()`

---

sfaR_summary   *Summarize a stochastic frontier model fitted with sfaR*

Description

Builds a tidy coefficient-and-test table from a fitted stochastic frontier model (typically from **sfaR**). The output combines:

- Maximum likelihood estimates of the frontier and auxiliary parameters (coefficients, standard errors, z-values, p-values), and

- Model-level diagnostics and tests, including variance components, skewness tests for OLS residuals, and a likelihood-ratio test for the presence of inefficiency.

Usage

```
sfaR_summary(fit)
```

Arguments

fit   A fitted stochastic frontier model object from **sfaR**. The object must support
                 summary() and contain elements such as mlLoglik, olsSkew, olsM3Okay, CoelliM3Test,
                 and AgostinoTest (as produced by **sfaR** fits).

Details

This function is intended as a convenient post-estimation summarizer whose output can be merged with other model summaries or used directly in tables and figures.

The function proceeds in several steps:

1. Extracts the maximum-likelihood coefficient table from summary(fit)$mlRes and renames columns to Estimate, StdError, Zvalue, Pvalue.

2. Constructs a one-row data.frame (TEST) containing model-level statistics returned by summary(fit), such as: Nobs, nXvar, nuZUvar, nvZVvar, mlLoglik, AIC, BIC, HQIC, sigmavSq, sigmauSq, Varu, Eu, and Expu. From these, it computes:

   - Sigma = sqrt(sigmauSq + sigmavSq), and
   - Gamma = sigmauSq / (Sigma^2).

   It also overwrites mlLoglik with fit$mlLoglik, adds OLS skewness (olsSkew), a flag for the expected skewness (olsM3Okay), and reshapes everything into a CoefName/Estimate table with Pvalue = NA.

3. Appends additional tests:

   Coelli's M3 test Stored as CoefName = "CoelliM3Test" with statistic and p-value taken from fit$CoelliM3Test.

   D'Agostino tests Three rows are added for "AgostinoOmn", "AgostinoSkw", and "AgostinoKrt", using fit$AgostinoTest@test$statistic and fit$AgostinoTest@test$p.value.

   LR test of inefficiency One row with CoefName = "LRInef", where the test statistic is summary(fit)$chisq. The p-value is computed using a chi-bar-square distribution via emdbook::qchibarsq() at 0.90, 0.95, and 0.99 quantiles, and then discretized to 0.10, 0.05, 0.01, or 1.

Finally, the ML coefficient table and the TEST rows are stacked (using data.table::rbindlist()) into a single data.frame with consistent columns.

Value

A data.frame with at least the following columns:

- CoefName: Name of the parameter or diagnostic/statistic.
- Estimate: Point estimate or test statistic.
- StdError: Standard error (for ML coefficients; NA for many diagnostics).
- Zvalue: z-statistic for ML coefficients (NA otherwise).
- Pvalue: p-value for ML coefficients and tests when available (NA if not applicable).

See Also

Other frontier analysis: equation_editor(), fit_organizer(), msf_workhorse(), sf_functional_forms(), sf_model_specifications(), sf_workhorse()

---

sf_functional_forms *Generate Functional and Distribution Forms for MSF Models*

Description

Builds algebraic strings for common production-function specifications and assembles a catalog of distributional assumptions for one-sided inefficiency terms used in stochastic frontier / multi-stage frontier (MSF) analysis.

Usage

```
sf_functional_forms(number_of_inputs = 5, include_trend = FALSE)
```

Arguments

number_of_inputs

> Integer. Number of input variables to enumerate in the functional forms (creates symbols `I1..Ik` and `lnI1..lnIk`). Default 5.

include_trend Logical. If `TRUE`, adjusts the Transcendental production form (TP) by dropping the last linear input term. See *Note* regarding when TP is present. Default `FALSE`.

Details

Functional forms (returned as character strings):

- `CD`: Cobb-Douglas - sum of logs of inputs, e.g., `lnI1 + lnI2 + ... + lnIk`.

- `TL`: Translog - CD plus all second-order log terms (squares and cross-products), e.g. %

- `LN`: Linear (commented out in current code). %

- `QD`: Quadratic (commented out). %

- `GP`: Generalized (commented out). %

- `TP`: Transcendental (commented out; see Note).

Terms are generated programmatically for `k = number_of_inputs`, with: `Ii` denoting input levels and `lnIi` their logarithms.

Distributional forms (list of lists): Each entry is named and contains `list(name, scaling = <logical>)`. Included options: `hnormal, tnormal, exponential, tslaplace, genexponential, tnormal_scaled, rayleigh, uniform, gamma, lognormal, weibull`. The `scaling` flag indicates whether the Wang-Schmidt (2002) scaling property is used in that specification.

Value

A `list` with two components:

- `fxnforms`: named list of character strings for functional forms (e.g., `$CD`, `$TL`). Forms that are commented out in the implementation will not appear.

- `distforms`: named list of distribution specifications; each is a list with elements `[1]` = distribution name and `scaling` = logical.

Note

The current implementation defines `CD` and `TL` and removes NULL entries before returning. Other forms (LN, QD, GP, TP) are commented out. If you plan to set `include_trend = TRUE`, ensure TP is actually included in `fxnforms`; otherwise, modifying `fxnforms$TP` will not have any effect (and would error if TP is not defined).

See Also

Other frontier analysis: [equation_editor()](), [fit_organizer()](), [msf_workhorse()](), [sf_model_specifications()](), [sf_workhorse()](), [sfaR_summary()]()

```
sf_model_specifications
```
Build Model Specifications

Description

Construct a specification table for frontier analysis by combining alternative production-function forms and distributional assumptions with technology choices and disaggregation levels. The function first creates a pooled grid of functional forms (`fxnforms`) and distributions (`distforms`), then augments it with crop-specific and demographic splits, and finally expands technology choices.

Usage

```
sf_model_specifications(
  data,
  distforms = sf_functional_forms()$distforms,
  fxnforms = sf_functional_forms()$fxnforms,
  technology_variables,
  mainF = 2,
  mainD = 1,
  demographic_variables = c("Female", "Region", "Ecozon", "EduCat", "EduLevel", "AgeCat"),
  crop_list = c("Beans", "Cassava", "Cocoa", "Cocoyam", "Maize", "Millet", "Okra",
    "Palm", "Peanut", "Pepper", "Plantain", "Rice", "Sorghum", "Tomatoe", "Yam")
)
```

Arguments

`data`	data.frame or data.table. A dataset that (ideally) contains the columns named in `demographic_variables`. **Note:** The current implementation references a global object `DATA` (not `data`) when deriving demographic levels; ensure `DATA` exists with those columns or adapt your environment accordingly.
`distforms`	named `list` of distributional forms. If `NULL`, defaults to `functional_forms()$distforms`.
`fxnforms`	named `list` of functional forms. If `NULL`, defaults to `functional_forms()$fxnforms`.
`technology_variables`	
	character vector of technology variables. The first element is taken as the default/primary technology.
`mainF`	integer index of the preferred functional form (in `fxnforms`). Default is 2.
`mainD`	integer index of the preferred distribution (in `distforms`). Default is 1.
`demographic_variables`	
	character vector of column names used for demographic disaggregation. Defaults to c("Female","Region","Ecozon","EduCat","EduLevel","AgeCat").
`crop_list`	character vector of crop names for crop-level disaggregation. Defaults to a selection of common crops.

Details

Procedure (high level):

1. Build a pooled grid of all `fxnforms` * `distforms`.

2. Keep rows matching `mainF` and `mainD` and tag pooled specs.

3. Add crop-specific rows for `crop_list` (using the main form/dist).

4. Add demographic splits for each variable in `demographic_variables`, using unique levels in `DATA[, var]` (see note in *Warning*).

5. Set `TechVar` to the first element of `technology_variables`; if additional technology variables are provided, add pooled rows for them with the main form/dist.

6. Reorder to prioritize `mainD` and create an `matching_type` flag with values `"fullset"` and `"optimal"`.

7. Remove demographic rows for the default demographic list and retain only `level == "Pooled"` in the final output.

Returned columns (final table):

- `disasg` - disaggregation variable (e.g., `"CropID"`).

- `level` - disaggregation level (e.g., `"Pooled"` or a crop/level).

- `TechVar` - selected technology variable.

- `f` - index of functional form (position in `fxnforms`).

- `d` - index of distribution (position in `distforms`).

- `matching_type` - marker column (`"fullset"` / `"optimal"`).

Value

A `data.frame` (data.table-compatible) of model specifications with columns `disasg`, `level`, `TechVar`, `f`, `d`, and `matching_type`.

Warning

The function calls `unique(DATA[, w])` inside a `tryCatch`; this requires a global object `DATA` containing the demographic columns. If `DATA` is absent or lacks a given variable, that split is skipped.

Assumptions

- `technology_variables` is non-empty; its first element is the default.

- `mainF` and `mainD` are valid indices within `fxnforms` and `distforms`, respectively.

See Also

Other frontier analysis: [equation_editor](), [fit_organizer](), [msf_workhorse](), [sf_functional_forms](), [sf_workhorse](), [sfaR_summary]()

---

sf_workhorse   *Stochastic frontier workhorse (unrestricted and shape-constrained)*

Description

Fits a stochastic frontier model for a given production specification and, when monotonicity is sufficiently violated, re-estimates a shape-constrained (restricted) frontier using minimum-distance methods. The function wraps the underlying `sfaR` estimation routines and a set helper utilities (e.g., `sf_functional_forms()`, `equation_editor()`, `fit_organizer()`, `translogEla()`, curvature/monotonicity checks) to produce:

1. Unrestricted stochastic frontier estimates and diagnostics.
2. Shape-constrained estimates that enforce monotonicity (and related regularity conditions) when needed.
3. Observation-level efficiencies, fitted values, and risk measures.
4. Observation-level elasticities, including a "returns-to-scale" term.

Usage

```
sf_workhorse(
  data,
  output_variable,
  input_variables,
  weight_variable = NULL,
  d,
  f,
  production_slope_shifters = NULL,
  intercept_shifters = NULL,
  inefficiency_covariates = NULL,
  risk_covariates = NULL,
  identifiers,
  include_trend = FALSE
)
```

Arguments

`data`	A data.frame or data.table containing all variables required for estimation, including the dependent variable `output_variable`, inputs `input_variables`, the weight variable `weight_variable` (if used), the unique ID variables listed in `identifiers`, and any variables referenced in `production_slope_shifters`, `intercept_shifters`, `inefficiency_covariates`, or `risk_covariates`.
`output_variable`	Character scalar. Name of the dependent/output variable in `data` used for the production frontier.
`input_variables`	Character vector of input variable names (e.g., land, labor, capital). The order of variables in `input_variables` determines how they map into the generic input labels `I1`, `I2`, ... used in the functional-form utilities.
`weight_variable`	Optional character scalar. Name of the sampling/observation weight variable. If `NULL`, all observations are given unit weight.

d	Distributional form index for the inefficiency term. This is passed to `sf_functional_forms()` and ultimately to `sfacross()` (e.g., to select half-normal, truncated-normal, etc.). The exact mapping is determined by `sf_functional_forms()`.
f	Functional form index for the production frontier (e.g., Cobb-Douglas, translog, quadratic), used to select from the list returned by `sf_functional_forms()`. The name of the chosen form (e.g., `"CD"`, `"TL"`, `"QD"`, `"LN"`) influences logging of variables and regularity checks.
production_slope_shifters	Character scalar giving the name of a slope-shifter variable in `data` (e.g., technology shift, policy dummy). Defaults to `NULL` for no slope shifter and is passed to `equation_editor()`.
intercept_shifters	Optional named list of intercept shifter variables for the production function. Typical structure: `list(scalar_variables = c(...), factor_variables = c(...))`, where the specific interpretation is handled inside `equation_editor()`.
inefficiency_covariates	Optional named list describing the inefficiency-function covariates. Typical structure: `list(scalar_variables = c(...), factor_variables = c(...))`, passed as `uhet` and `muhet` to `sfacross()` when non-`NULL`.
risk_covariates	Optional named list describing the production-risk (noise) covariates. When non-`NULL`, used as `vhet` in `sfacross()` to allow heteroskedasticity of the noise term.
identifiers	Character vector of variable names that uniquely identify observations (e.g., `c("unique_identifier","Survey","CropID","HhId","EaId","Mid")`). These IDs are carried through to efficiency, elasticity, and risk outputs.
include_trend	Logical; if `TRUE`, the last element in `input_variables` is treated as a trend/technology variable and handled accordingly in the functional-form and elasticity calculations. Defaults to `FALSE`.

Details

The function proceeds in several steps:

1. **Setup and functional form selection:** Using `sf_functional_forms()`, the function determines the appropriate production functional form and distributional specification based on the number of inputs (`number_of_inputs`), the `f` and `d` indices, and the `include_trend` flag. This includes whether the model is specified in logs (e.g., CD/TL/GP/TP) or levels.

2. **Variable construction and equation building:** Inputs in `input_variables` are mapped to generic labels (`I1`, `I2`, ...) and their log transforms are created when needed. The outcome variable is set to `Y` or `lnY`. The function then calls `equation_editor()` to construct the production, inefficiency, and risk equations used by `sfacross()`.

3. **Unrestricted stochastic frontier estimation:** The function iterates over a grid of optimization methods (`nr`, `nm`, `bfgs`, `bhhh`, ...) and tolerance settings, calling `sfacross()` until a successful convergence is reported. It extracts observation-level efficiencies via `sfaR::efficiencies()`, constructs fitted values, and builds an observation-level risk metric based on squared deviations from mean output.

4. **Elasticities and regularity checks:** Using `fit_organizer()` and `translogEla()`, the function computes elasticities and returns-to-scale-type measures, and evaluates monotonicity and curvature via `translogCheckMono()` and `translogCheckCurvature()` (or a coefficient-sign check for CD/LN forms).

5. **Shape-constrained frontier (if needed):** If the monotonicity measure mono falls below 0.80, the function constructs a set of linear restrictions using translogMonoRestr() and solves a quadratic programming problem (via quadprog::solve.QP and fall-back matrix inversions using **Matrix**, **MASS**, **corpcor**) to obtain constrained coefficients. For TL/QD forms a constrained frontier (lcFitted) is computed with translogCalc(); for CD/LN forms, constrained coefficients are obtained via a SEM representation using **lavaan**.

6. **Re-estimation under constraints:** Given the constrained frontier, the function rebuilds the production equation with equation_editor() and re-estimates a stochastic frontier (sfc) using the same optimization grid. Constrained efficiencies, risks, and elasticities are computed and summarized, and regularity checks are repeated.

Unrestricted and restricted summaries are then stacked, with a restrict flag ("Unrestricted" vs. "Restricted"), for downstream comparison.

Value

A named list with four data.frames:

- sf: Coefficient-level results combining unrestricted and, when applicable, restricted SFA estimates, plus monotonicity/curvature diagnostics. Columns include (at minimum) CoefName, Estimate, StdError, Zvalue, Pvalue, and restrict.

- ef: Observation-level efficiency results (unrestricted and restricted), including the ID variables in identifiers, weights, efficiency measures (e.g., u), model-fitted values (mlFitted), and restrict.

- el: Observation-level elasticities and returns-to-scale-type measures. Includes IDs, weights, elasticity columns (e.g., el1, el2, . . ., and the summed elasticity), and restrict.

- rk: Observation-level risk measures (unrestricted and restricted), including IDs, weights, and risk, plus restrict.

See Also

Other frontier analysis: equation_editor(), fit_organizer(), msf_workhorse(), sf_functional_forms(), sf_model_specifications(), sfaR_summary()

---

study_setup   *Initialize Study Environment and Directory Structure*

Description

Initialize Study Environment and Directory Structure

Usage

```
study_setup(
  myseed = 1980632,
  project_name,
  local_directories = list(home = file.path("replications", project_name), output =
    file.path("replications", project_name, "output"), matching =
  file.path("replications", project_name, "output", "matching"), treatment_effects =
  file.path("replications", project_name, "output", "treatment_effects"), estimations =
  file.path("replications", project_name, "output", "estimations"), figure_data =
```

```
        file.path("replications", project_name, "output", "figure_data"), figure =
        file.path("replications", project_name, "output", "figure"))
)
```

Arguments

myseed	Numeric/integer scalar seed. Will be coerced to integer. Default: 1980632.
project_name	Length-1, non-NA character project name (required).
local_directories	
	Named list of character paths to create. Defaults use `project_name`.

Value

List with wd (directories) and seed.

---

treatment_effect_calculation

Compute log-linear treatment effects (ATE/ATET/ATEU) for multiple outcomes

---

Description

For a given matching specification index `i`, this function:

1. Loads the corresponding matched weights file from `matching_output_directory`,

2. Merges those weights into the analysis `data` via `unique_identifier`,

3. Optionally normalizes each outcome by the first element of `outcome_variables` (e.g., to construct per-hectare or per-unit measures),

4. Fits a weighted log-linear model for each outcome with interactions between `Treat` and the supplied matching covariate formulas,

5. Transforms fitted differences into percentage treatment effects, trims extreme values, and computes weighted ATE, ATET, and ATEU alongside model fit statistics.

Usage

```
treatment_effect_calculation(
  data,
  outcome_variables,
  normalize = NULL,
  i,
  match_specifications,
  matching_output_directory,
  match_formulas
)
```

Arguments

data
: A data.frame containing at least:

  - unique_identifier: unit identifier used for merging matched weights,
  - Treat: treatment indicator (logical or 0/1),
  - Area: plot size (positive; used for input checks and typical scaling),
  - all variables named in outcome_variables,
  - any covariates referenced in match_formulas$general_match and match_formulas$exact_ma

outcome_variables
: Character vector of outcome variable names (e.g., c("HrvstKg","Area","SeedKg","HHLaborAE","

normalize
: Logical scalar. If TRUE, each outcome in outcome_variables is divided by the first element of outcome_variables in the merged data (e.g., to obtain outcomes per hectare or per unit).

i
: Integer index selecting the row of match_specifications and the corresponding matching result file (zero-padded via ARRAY to "NNNN.rds").

match_specifications
: A data.frame with at least a column ARRAY. The i-th row is used to locate the matching output file and is cbind-ed to the returned results.

matching_output_directory
: Directory containing per-specification matching results named "match_0001.rds", "match_0002.rds", etc. Each RDS is expected to contain an element $md with columns unique_identifier and weights.

match_formulas
: A list with:

  - general_match: a formula; the RHS is accessed via as.character(...)[3],
  - exact_match: a formula; the RHS is accessed via as.character(...)[2].

Details

For each outcome oc in outcome_variables, the function:

1. Filters to observations with non-missing Treat, the outcome, and weights,

2. Constructs a log-linear model of the form

   log(oc + eps) ~ Treat * (general_match_rhs + exact_match_rhs)

   where eps = min(oc[oc > 0]) * 1/100 in the estimation sample,

3. Fits the model using lm() with weights given by the matched weights,

4. Predicts fitted log outcomes for treated (Treat = 1) and untreated (Treat = 0) counterfactuals,

5. Computes individual percentage treatment effects

$$TE_OLS = \{\exp(\hat{y}_1 - \hat{y}_0) - 1\} \times 100,$$

   trims them to the interval $[-100, 100]$, and then forms:

   - ATE: weighted mean of TE_OLS over all units,
   - ATET: weighted mean of TE_OLS among treated units,
   - ATEU: weighted mean of TE_OLS among untreated units.

In addition, for each outcome the function records model diagnostics: AIC, log-likelihood, R^2, adjusted R^2, F-statistic, and the sample size used in the regression.

Value

A data.frame (atet_scalar) with the columns from match_specifications[i,] repeated across rows, and the additional columns:

- outcome: outcome variable name,
- treatment: name of the treatment indicator (always "Treat"),
- level: one of "ATE", "ATET", "ATEU", "aic", "ll", "R2", "N", "Ft", "R2a",
- est: numeric estimate corresponding to level.

One block of rows is returned per outcome in outcome_variables. If a model fails for a given outcome, that outcome is skipped.

See Also

Other matching: covariate_balance(), draw_matched_samples(), match_sample_specifications(), treatment_effect_summary(), write_match_formulas()

---

treatment_effect_summary

Summarize Treatment Effect Estimates Across Matching Specifications

---

Description

Aggregates and summarizes treatment effect results stored as .rds files in a specified output directory. The function reads each RDS file, combines all results into a single data frame, filters out invalid estimates, and computes jackknife-style summary statistics (mean, standard error, sample size, t-value, and p-value) by matching specification and outcome.

Usage

```
treatment_effect_summary(treatment_effects_output_directory)
```

Arguments

treatment_effects_output_directory

Character string giving the path to the directory containing .rds files with treatment effect results (e.g., outputs of treatment_effect_calculation() saved across specifications).

Details

The function proceeds through the following steps:

1. Lists and reads all .rds files in treatment_effects_output_directory.
2. Combines them into a single long-format data frame using data.table::rbindlist().
3. Removes rows with invalid estimates (NA, NaN, Inf, or -Inf).
4. Separates base (non-bootstrap) results where boot == 0 and joins these with jackknife summary statistics computed via doBy::summaryBy() across method, distance, link, outcome, and level.
5. Renames summary columns to:

- jack_mean: mean estimate
- jack_se: standard error
- jack_n: sample size (number of replicates)
- jack_tvl: t-value (jack_mean / jack_se)
- jack_pvl: two-sided p-value

Value

A data.frame containing summarized treatment effect estimates with columns:

- method, distance, link, outcome, level
- est (original estimate for non-bootstrap sample)
- jack_mean, jack_se, jack_n, jack_tvl, jack_pvl

See Also

Other matching: covariate_balance(), draw_matched_samples(), match_sample_specifications(), treatment_effect_calculation(), write_match_formulas()

---

write_match_formulas   *Construct Matching Formulas for Exact and General Matching*

Description

Builds two types of matching formulas commonly used in propensity score or covariate matching workflows:

- An **exact match** formula specifying categorical variables to match exactly.
- A **general match** formula specifying numeric and factor covariates for distance-based matching.

The function returns both formulas as character strings that can be used in matching functions such as MatchIt::matchit() or Matching::Match().

Usage

```
write_match_formulas(
  match_variables_exact,
  match_variables_scaler,
  match_variables_factor
)
```

Arguments

match_variables_exact

> A character vector of variable names to be included in the exact match component (e.g., "gender", "region").

match_variables_scaler

> A character vector of continuous or scalar variable names to include as covariates in the general match formula (e.g., "age", "income").

match_variables_factor

> A character vector of factor variable names to be included as dummy-coded categorical covariates in the general match formula.

Value

A named list with two elements:

exact_match A string representing the exact match formula (factor variables only).

general_match A string representing the general match formula, including both continuous and factor covariates on the right-hand side of `Treat ~`.

See Also

Other matching: covariate_balance(), draw_matched_samples(), match_sample_specifications(), treatment_effect_calculation(), treatment_effect_summary()

Index